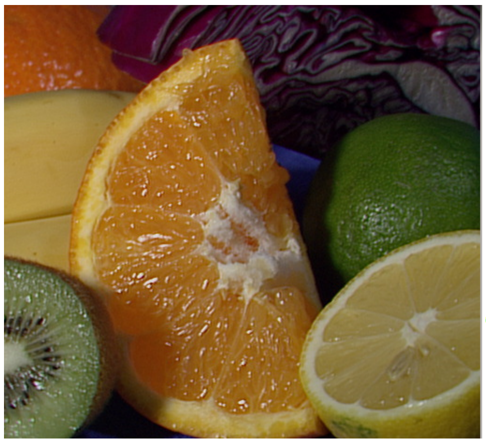
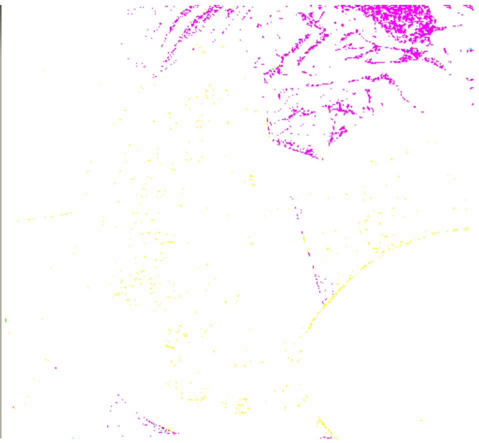
OpenCV using k-means to posterize an image

I want to posterize an image with k-means and OpenCV in C++ interface (cv namespace) and I get weird results. I need it for reduce some noise. This is my code:

```
#include "cv.h"
#include "highgui.h"

using namespace cv;

int main() {
    Mat imageBGR, imageHSV, planeH, planeS, planeV;

    imageBGR = imread("fruits.jpg");
    imshow("original", imageBGR);

    cv::Mat labels, data;
    cv::Mat centers(8, 1, CV_32FC1);
    imageBGR.convertTo(data, CV_32F);

    cv::kmeans(data, 8, labels,
            cv::TermCriteria(CV_TERMCRIT_ITER, 10, 1.0),
            3, cv::KMEANS_PP_CENTERS, &centers);
    imshow("posterized hue", data);
    data.convertTo(data, CV_32FC3);

    waitKey();
    return 0;
}
```

But I get a weird result

First image: original

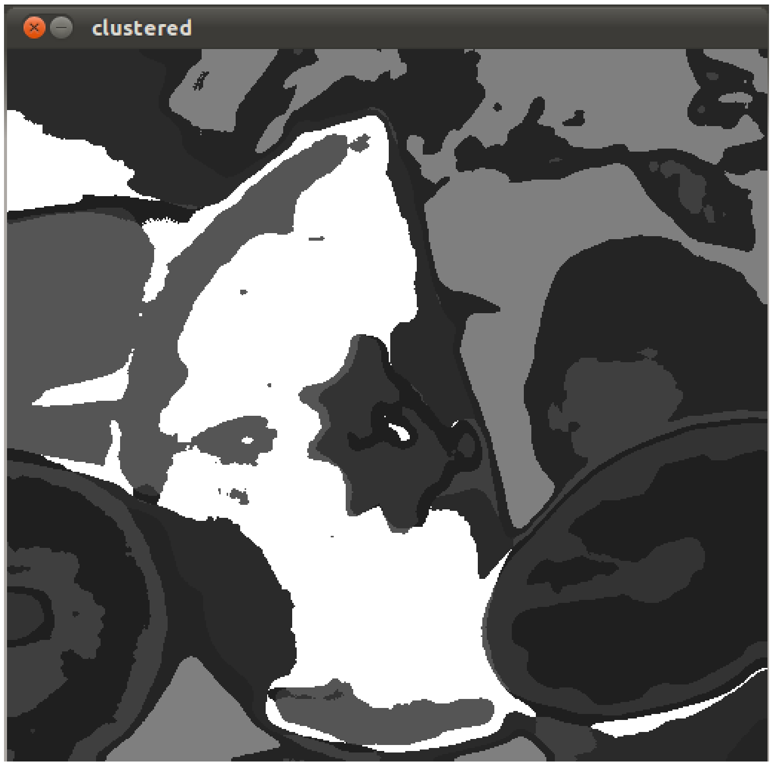
Second image: after k-means.

Any advice?

Update: the right solution. maybe someone can help me in optimize the code?

```
    cv::split(src, bgr);
    // i think there is a better way to split pixel bgr color
    for(int i=0; i<src.cols*src.rows; i++) {
        p.at<float>(i,0) = (i/src.cols) / src.rows;
        p.at<float>(i,1) = (i%src.cols) / src.cols;
        p.at<float>(i,2) = bgr[0].data[i] / 255.0;
        p.at<float>(i,3) = bgr[1].data[i] / 255.0;
        p.at<float>(i,4) = bgr[2].data[i] / 255.0;
    }

    int K = 8;
    cv::kmeans(p, K, bestLabels,
            TermCriteria( CV_TERMCRIT_EPS+CV_TERMCRIT_ITER, 10, 1.0),
            3, KMEANS_PP_CENTERS, centers);

    int colors[K];
    for(int i=0; i<K; i++) {
        colors[i] = 255/(i+1);
    }
    // i think there is a better way to do this mayebe some Mat::reshape?
    clustered = Mat(src.rows, src.cols, CV_32F);
    for(int i=0; i<src.cols*src.rows; i++) {
        clustered.at<float>(i/src.cols, i%src.cols) = (float)(colors[bestLabels.
//      cout << bestLabels.at<int>(0,i) << " " <<
//              colors[bestLabels.at<int>(0,i)] << " " <<
//              clustered.at<float>(i/src.cols, i%src.cols) << " " <<
//              endl;
    }

    clustered.convertTo(clustered, CV_8U);
    imshow("clustered", clustered);

    waitKey();
    return 0;
}
```

Result:



c++ image-processing opencv k-means noise-reduction

edited Jul 10 '13 at 2:20
Bill the Lizard ♦
140k 97 355 630

asked Mar 5 '12 at 23:22
nkint
2,230 23 59

It could just be that you need more iterations and/or a smaller epsilon. I'd suggest you try removing "CV_TERMCRIT_EPS" for now, and play with the number of iterations in your TermCriteria. See if that helps. – jilles de wit Mar 6 '12 at 9:22

i'm just approaching computer vision, image processing and machine learning but for me there is another mistake in what i'm doing and not only a parameter fine tuning.. – nkint Mar 6 '12 at 12:19

I'm not suggesting you start parameter tuning, I'm suggesting you simplify your code to test that what you're trying works in its most basic form. Removing epsilon and increasing the number of iterations removes unneeded trickery. – jilles de wit Mar 6 '12 at 12:31

1 ok i've redone it from skretch and edit my question – nkint Mar 7 '12 at 19:47

1 see edit now it is a good result mayebe it should be optimized – nkint Mar 24 '12 at 14:34

show **3** more comments

2 Answers

I am no expert in OpenCV so I will give a general advice that relates to your question K-means takes list of vectors which is essentially a matrix:

```
[x0, y0, r0, g0, b0]
[x1, y1, r1, g1, b1]
[x2, y2, r2, g2, b2]
.
.
.
```

You are giving it an image which is not going to work. You first have to convert the image to this k-means matrix format. For each pixel of the source image you have one row in the resulting matrix. Also note that you should scale the values so that they all have similar values. If you don't do that, x and y coordinates will usually have much higher "gravity" than the color which leads to unsatisfactory results.

C++ pseudocode:

```
int pixel_index = 0;
for (int y = 0; y < image height; y++)  {
  for (int x = 0; x < image width; x++)  {
     matrix[pixel_index][0] = (float)x / image width;
     matrix[pixel_index][1] = (float)y / image height;
     matrix[pixel_index][2] = (float)pixel(x, y).r / 255.0f;
     matrix[pixel_index][3] = (float)pixel(x, y).g / 255.0f;
     matrix[pixel_index][4] = (float)pixel(x, y).b / 255.0f;
  }
}
// Pass the matrix to kmeans...
```

As a result, you get labels of each individual pixel which corresponds to the cluster it has been assigned to. You then need to determine the color of the clusters - this can vary from taking the center pixel color value to computing an average/median color of the cluster. After you determine the color, just walk the image and set pixels to their cluster colors:

```
for (int y = 0; y < image height; y++)  {
  for (int x = 0; x < image width; x++)  {
     int index = y * image width + x;  // This corresponds to pixel_index above
     int cluster_index = labels[index]; // 0 to 7 in your case
     Color color = colors[cluster_index];  // Colors is an array of 8 colors of th
     image.setpixel(x, y, color)
  }
}
```

If you prefer to use HSV instead of RGB, just use HSV values instead of RGB ones.

It is possible that OpenCV has functions that perform exactly the conversion I described above but I was unable to quick find them using Google.

answered Mar 7 '12 at 20:26
Karel Petranek
10.4k 2 19 43

sorry but where can i find infos about this kmeans specific input format? – nkint Mar 7 '12 at 22:46

In the OpenCV documentation (opencv.willowgarage.com/documentation/cpp/…): `samples – Floating-point matrix of input samples, one row per sample` where sample means a multi dimensional point. In the case of a color image, the point has 5 dimensions (x, y, r, g, b). This is pretty much the standard way to do kmeans, OpenCV just expresses it using its own data structures. For a general kmeans introduction I recommend the Machine learning videos about kmeans at ml-class.org. – Karel Petranek Mar 7 '12 at 23:08

i am already subscribed the next course has to begin yet! :) – nkint Mar 7 '12 at 23:37

it works perfectly thanks. see edits i've posted the right code. mayebe it could be optimized with some opencv api methods that i don't know – nkint Mar 24 '12 at 14:35

I'm glad it works :) Your code has a subtle bug though - you are assigning all three channels to the 0th element in the first loop. That means you are clustering only by color (not coordinates) as they get overwritten. Also make sure you use floating point division when calculating the normalized x, y coordinates. – Karel Petranek Mar 25 '12 at 11:48

add comment

If you don't need to x,y coordinates in your k-means, you can arrange the data much quicker as follows using the reshape command:

```
int origRows = img.rows;
notes << "original image is: " << img.rows << "x" << img.cols << endl;
Mat colVec = img.reshape(1, img.rows*img.cols); // change to a Nx3 column vector
cout << "colVec is of size: " << colVec.rows << "x" << colVec.cols << endl;
Mat colVecD, bestLabels, centers, clustered;
int attempts = 5;
int clusts = 8;
double eps = 0.001;
colVec.convertTo(colVecD, CV_32FC3, 1.0/255.0); // convert to floating point
double compactness = kmeans(colVecD, clusts, bestLabels,
      TermCriteria(CV_TERMCRIT_EPS+CV_TERMCRIT_ITER, attempts, eps),
      attempts, KMEANS_PP_CENTERS, centers);
Mat labelsImg = bestLabels.reshape(1, origRows); // single channel image of labe
cout << "Compactness = " << compactness << endl;
```

answered Jun 28 '12 at 14:44

zzzz
184 2 12

good! nice way, i was looking for an easy way to do it! thanks! – nkint Jun 28 '12 at 15:51

add comment

Not the answer you're looking for? Browse other questions tagged c++ image-processing opencv k-means noise-reduction or ask your own question.